

2

AD-A264 839



NOTATION PAGE

Form Approved
OMB No. 0704-0188

RT DATE

3 REPORT TYPE AND DATES COVERED
FINAL 1 Mar 89 -30 Apr 92

4. TITLE AND SUBTITLE

"DESIGN ISSUES FOR HIGH PERFORMANCE ENGINEERING
INFORMATION SYSTEMS" (U)

5 FUNDING NUMBERS

61102F
2304/A2

6 AUTHOR(S)

Drs. Nick Roussopoulos, Timos Sellis, Leo Mark and
Christos Faloutsos

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Maryland
Computer Science
College Park MD 20742

8. PERFORMING ORGANIZATION
REPORT NUMBER

AFOSR-TR

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFOSR/NM
110 Duncan Ave Suite B115
Bolling AFB DC 20332-0001

10. SPONSORING/MONITORING
AGENCY REPORT NUMBER

AFOSR-89-0303

11. SUPPLEMENTARY NOTES

93-10747

12a DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release;
Distribution unlimited



UL

93 5 13 044

13. ABSTRACT (Maximum 200 words)

It is increasingly being recognized that an Engineering Information System will be the fundamental component of any design and manufacturing system in the future and that all components in such a system have to be engineered around the management and control of information. Commercially available database systems do not meet the information and processing needs of design and manufacturing environments, consequently, extensions of database systems are necessary to realize Engineering Information Systems. These systems will support multimedia databases of significant size and complexity, and, therefore, one of the most important issues is performance. Addressed in this project were the computational and architectural aspects of EIS and rule management techniques for maintaining consistency. The studies produced promising models and solutions.

14. SUBJECT TERMS

15. NUMBER OF PAGES

20

16. PRICE CODE

17. SECURITY CLASSIFICATION
OF REPORT

UNCLASSIFIED

18. SECURITY CLASSIFICATION
OF THIS PAGE

UNCLASSIFIED

19. SECURITY CLASSIFICATION
OF ABSTRACT

UNCLASSIFIED

20. LIMITATION OF ABSTRACT

SAR

Final Report — March 1992

Project Title: Design Issues for High Performance Engineering Information Systems

Grant Number AFOSR-89-0303

Principal Investigator: *Nick Roussopoulos*

Co-Principal Investigators: *Timos Sellis, Leo Mark, Christos Faloutsos*

Department of Computer Science, University of Maryland

College Park, MD 20742

DISPATCHED 5

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

Executive Summary

It is increasingly being recognized that an *Engineering Information System* will be the fundamental component of any design and manufacturing system in the future and that all components in such a system have to be engineered around the management and control of information. Commercially available database systems do not meet the information and processing needs of design and manufacturing environments; consequently, extensions of database systems are necessary to realize Engineering Information Systems. These systems will support multimedia databases of significant size and complexity, and, therefore, one of the most important issues is performance.

In this project, we addressed the computational and architectural aspects of Engineering Information Systems and rule management techniques for maintaining consistency. Our studies produced promising models and solutions. These are summarized as follows:

- Incremental computation models are needed to achieve the required performance in client-server architectures, time and version management, rule execution, and image retrieval.
- Client-server architectures are needed to provide access to multiple heterogeneous and autonomous databases while maintaining high availability and low communication overhead;
- Efficient organization, management and execution of large rule bases are needed to make the Engineering Information Systems active and to maintain consistency amongst the voluminous and overlapping databases.
- Multidimensional indexing is needed for efficient retrieval of images, text, and complex engineering objects.

This report reviews the most important results of our research on Engineering Information Systems funded through this AFOSR contract and a companion contract with the National Science Foundation. The results have been published in refereed journals, refereed conferences, and University of Maryland Technical Reports, and Theses. These are divided as follows: 11 journal publications, 4 submissions for publication, 7 conference publications, 8 Technical Reports, 4 PhD dissertations and 8 Masters Theses.

1. PART A: Motivation and Requirements

Part A of this report motivates the research and derives the requirements for High Performance Engineering Information Systems.

1.1. Engineering Information Systems

Up until now, data has been treated in a passive way, namely collected and stored during the operation and analyzed offline. This passive data collection is only good for post-mortem analysis, not for controlling and driving the operation. It is, however, increasingly being recognized that the lift-off of the space shuttle is not just the physical movement per se, nor the static snapshot of the data at any time instance, but the information about all the events preceding the lift-off, the controlled decisions that drive it, and the effects of those decisions on its future states. The physical movement is the result of a set of controlled actions, and is as important, or unimportant if you like, as the rotating movement of the wheels of your car during your search for the right exit off a highway in New York City.

It is becoming better understood that *Engineering Information Systems (EIS)* will be the most important component in the design and operation of complex systems, and that all other components of these systems have to be engineered around the management and control of information. Only the realization of the importance of information engineering and the appropriate budgeting of an Engineering Information System in the cost of a system will bring us into the information age and give us a chance to contend with our far-east competitors in high-technology design and manufacturing.

Recently, there has been considerable interest in providing a framework for information sharing and exchange in engineering environments. *Engineering Information Systems (EIS)* were, and are, the focus of several government sponsored research projects. These projects try to consolidate and integrate the engineering support environments that have been and are being developed to support planning, design automation, manufacturing, resource management, etc. Along this direction, there has already been some work in extending existing business-tailored database management systems to accommodate engineering applications. In particular, relational DBMSs storing their factual data in relations (or tables) have been used in support of Computer Aided Design (CAD), Computer Integrated Manufacturing (CIM), and Artificial Intelligence and Expert Systems. The main difference between the business applications and EISs lies in the kind of information the applications use. Business applications mainly deal with large volumes of *structured data* and have well understood access patterns; engineering applications usually involve sophisticated *control mechanisms*, deal with both structured and unstructured data, but have rather unpredictable access patterns. Therefore, an EIS should be able to support explicit representations of control information in addition to factual information.

Figure 1 shows an example of an EIS supporting a Computer Integrated Manufacturing system (CIM). The EIS integrates several subsystems, one for Manufacturing and Resource Planning (MRP), one for Computer-Aided Design (CAD), and one for Computer-Aided Process Planning (CAPP). The MRP system contains a variety of information, especially the bill of material (B.O.M.) describing part explosion and the **PART** information containing the individual part descriptions including version information. The CAD system contains **DESIGNS** and **DRAWINGS** in various versions. The CAPP system contains process **PLANS**, **ROUTING** information, etc.

Our example concentrates on the MRP and CAD systems. There are a number of situations where deductive rules are needed for retrieval of complex data stored in two or more of the subsystems. For example, the part-explosion (transitive closure) of the **B.O.M.** is useful for both the manufacturing and resource planning subsystems. In addition, there is a considerable amount of redundancy and dependency between and within the data stored by the two systems. Active rules are needed to specify how updates in one subsystem propagate to the others (change propagation). For example, data entry made in the **DRAWING** relation as a result of storing a graphical **DESIGN** should also be done in the **B.O.M.** and **PART** relations. Similarly, modifications in the **DRAWING** relation are made as a result of finalizing a **DESIGN**; entries may not be accepted into the **DRAWING** relation if parts used in a **DESIGN** are not present in the **PART** relation. Such dependencies between the redundant data require the use of mechanisms to maintain consistency, support data exchange, and enforce change propagation between the two systems.

1.2. Why Use Databases in EISs?

It is clear that an EIS will contain large volumes of data and, therefore, database technology is needed for providing data sharing, consistency and integrity, recovery, archival, access control, and all the other benefits of a controlled environment. Furthermore, a typical EIS environment is naturally distributed among a large number of tools and special purpose workstations, such as schematic analyzers, graphics editors, simulators, database management systems, etc. Most of the interactions will originate from a workstation or a tool tailored to some specific task. For example, a graphics workstation may be used in a CAD system for editing a schematic layout and a faster processor or a super computer for running its logic simulations. Distributed databases have dealt with distributed concurrency control protocols and the associated issues of data consistency and integrity. Therefore, EISs must utilize the existing database technology.

1.3. Database Challenges in EIS

Database technology, however, was developed to help business data processing applications which maintain static snapshots of the data. The dynamic aspects of an EIS environment, the evolution of data and events, and the heterogeneity of data, hardware, and software, present the biggest database challenges in EIS.

It is highly unlikely that all the functionality of special purpose processors and tools that currently exist on the market or those which will be developed in the near future can be provided by a single host. It is also highly unlikely, that all data will be integrated in a super-database repository accessible by everyone dealing with an EIS. Not mentioning the immeasurable political problems inhibiting such an integration, there are technological problems as well. First, the variety and diversity of data and software for EIS processing makes integration a utopia. Second, the cost of such a solution, even if it were technically feasible, would be astronomical. Third, experience has shown that cost effective solutions rely mainly on simple systems rather than on monolithic giants.

For all these reasons, the majority of the database community is leaning towards the idea of "interoperability" of autonomous and heterogeneous databases, as opposed to the dream of "total data integration." However, heterogeneous database interoperability implies enormous amounts of data translation, data transmission, environment switching overhead, cooperative software, and distributed control. The

technology for achieving these is in its infancy. New computation models and architectures are necessary to achieve the required performance. Similarly, new rule specification models for controlling the evolution of these autonomous databases are needed. These are the problems with which we have dealt in this project.

1.4. Incremental Computation Models

Conventional computation models are based on *re-execution*. That is, all computation is repeated each and every time results are needed. Nothing is retained from previous executions and, many times, even optimization of the computation is repeated. However, a lot of the cost can be saved by using more intelligent computation models which retain some of their results, or access paths to these results, in persistent storage for reuse. Our research in the last several years has centered around a new concept of *incremental computation models* which utilize *cached* results or access paths. These results are realized again by applying the computation on the input *differentials* as opposed to re-executing the computation on the whole and almost unchanged input. This concept is illustrated in Figure 2 where a "cross-reference" of the two data files R and S is computed only once from the whole R and S files, and, maintained thereafter incrementally using only the differential files δR and δS .

Incremental computation is performed on demand or periodically (using a lazy evaluation strategy) on small input increments. The cost of computing is amortized over the life-cycle of the computed information, a concept that is absolutely orthogonal to the re-execution of transient and non-persistent software we have been used to. Because computation is done on increments, performance is improved by several orders of magnitude.

Incremental computation models are the foundation of the research done in this project. The following subsections are all based on incremental computation models.

1.5. Enhanced Workstation Client-Server Database Architectures

Client-Server architectures originated in engineering application environments where data is mostly processed in powerful workstations with CAD/CAM or other special purpose software, while centralized repositories with check-in and check-out protocols are predominantly used for maintaining data consistency. These protocols were developed to work at the file level and, the clients could not submit queries or updates to these files on the servers. On the other hand, database Client-Server architectures were limited to providing query capability for the clients, but no update propagation protocols for staging the data. The main reason for this was the difficulty with the proprietary formats of the database vendors and their tendency to keep their system architectures closed.

After studying these two types of architectures, we proposed, designed, built and evaluated an Enhanced Client-Server (ECS) architecture which combines the features of both. First, clients have been enhanced to run from a workstation which offers query processing and local disks for data staging. Second, we applied incremental update propagation algorithms for maintaining the staged data consistent. Finally, we built incremental access methods for several commercial DBMSs to demonstrate the concept of interoperability. This architecture is depicted in Figure 3 with three commercial DBMSs and a number of workstation clients.

In the last year of this project, we designed and ran a set of simulation packages to evaluate the performance of three different Client-Server architectures. The simulations showed that a) the ECS architecture is superior in performance due to the utilization of the local disks which permit parallel access to the data and the incremental update propagation; Figure 4(a) illustrates the speed-up obtained in total transaction throughput over conventional Client-Server architectures, b) the ECS architecture is scalable and the only limiting factor in scaling up is the serialization of the updates that need to be synchronized. However, even in update intensive situations, the performance is at least an order of magnitude higher than the nearest competitor, and for relatively static databases, several orders of magnitude better. Figure 4(b) illustrates the scalability of the architectures with runs of up to 200 workstation clients attached to a single server. This performance cannot be replicated even in a multimillion super database server.

1.6. Incremental and Decremental Time and Version Management

A conventional database maintains only the current state of the modeled world. A transaction time database maintains not only the current state, but also all previous states of the world; see Figure 5. Efficient management of such temporal data that accumulates to enormous volumes is essential in providing version management in an EIS. We developed incremental and decremental computation models, and combined them with standard query optimization techniques to achieve efficient access to temporal data. In addition, we developed a query language that supports advanced queries on change behavior in temporal data. These two provide a foundation for monitoring the evolution of versions.

In order to support the above capabilities, we extended the databases to include a backlog which stores a complete history of time-stamped change requests. Previous states of a relation are computed by *time-slicing* its backlog and result in cached access paths; see Figure 6. A time-slice, $R(t)$, is computed *incrementally* from the closest cached time-slice, $R(t_y)$, where $t_y < t$ or *decrementally* from the closest cached time-slice, $R(t_x)$, where $t < t_x$, whichever is more efficient; see Figure 7.

In addition to using the backlogs for incremental and decremental computation of database states, the backlogs themselves can be made the subjects of queries. This allows new types of queries on database *change behavior*, especially important in EISs. For example, if component types $X_1, X_2, \dots, X_n$ of an object X are replaced when they fail to function properly and these replacements are recorded in the backlog, then the following questions can be answered by queries on the backlog:

- When was the most recent failure of a component type X_i of object X ?
- Which component type X_i of object X has the highest failure rate?
- Does any component type X_i of object X exhibit *unusual* failure behavior?

1.7. Active Rule Bases

Due to the large number of dependencies between redundant data in an EIS, an active rule subsystem is needed to maintain data consistency, support data exchange, and enforce change propagation between systems. Because of the complexity of an EIS environment, the rule subsystem must support a very large number of rules which themselves need be stored in a *rule base*. This rule base must itself be efficiently organized and accessed by the EIS.

Although rules can be written in any programming language, such an approach would neither allow any efficient retrieval of the rules to be *fired* in a given event, nor any discovery of rule-to-rule negative side-effects. To accommodate these requirements, we have developed a powerful specification language for engineering rules. This language can be used to specify *Update Dependencies* between data. An example of rule specification is given in Figure 8. The rule for deleting a **PART** has two alternatives. If the **PART** is in stock ($QTY > 0$), then the deletion is rejected because we want to keep the information about all parts that we have in stock. On the other hand, if the **PART** is not in stock ($QTY = 0$), then information about it is first deleted from the **STOCK** relation and then removed from the **PART** relation. Similarly, insertion in the **BOM** (Bills of Material) relation is only accepted if no cycles are discovered in the design. Furthermore, an insertion of a **PART** information must be accompanied by an insertion in the **BOM** relation.

The Update Dependency Language supports a wide variety of applications such as, walk-through guidance control systems, cause-effect systems, statistical information gathering, knowledge acquisition, database integrity enforcement, database view updates, policy enforcement, and production control. The language differs from other formalism for specifying database integrity because the designer uses it to specify the correct evolution of the database rather than the valid states of the database. This formalism has been successfully used in a number of applications (e.g., in the functional specification and implementation of an integrated CAD/CAPP/MRP system).

A rule succeeds when one of the right-hand-sides of the rule succeeds. A right-hand-side succeeds when its condition evaluates to true and all the implied rules and primitive operations succeed. To support the efficient execution of rules as an integral part of an EIS, we have developed new search and locking strategies for the Update Dependency Language.

1.8. Images, Rules and Text

The data types of commercial DBMSs are all alphanumeric. These systems do not adequately support other varieties of data types used in EISs, such as images, drawings, documents, and rules. Figure 9 shows several of these types as laid on a two-dimensional space: VLSI drawings, cartographic data, data correlation charts, and rule indexing. In each of these cases, the goal is to find all objects that are contained a two-dimensional window. Extending the search space to two or higher dimensions increases search complexity and makes queries extremely slow. In order to efficiently retrieve such data we use R-trees.

R-trees are based on the most popular one-dimensional access method, called *B-trees*, but are capable of indexing multidimensional space. Their most frequent application is in two-dimensional space where they were introduced for searching VLSI designs. In the past several years we have improved the performance of R-trees by providing fine-tuning techniques for space compaction and more effective utilization for a variety of spatial objects such as points, line segments, regions, and polygons.

Although spatial access methods are essential in EIS environments, they have not been extensively utilized because a) they require data file reorganization and reformatting so that these methods can be applied, and b) they cannot be incorporated with neither existing commercial systems which are totally closed, nor with in-house systems which are very hard to extend. Reorganizing existing data is totally

impractical, not solely because of the high conversion cost, but mainly because duplication of data leads to inconsistencies.

Our approach in dealing with this problem is similar to that of interoperability of database systems. We treat spatial data the same way as other data, that is, it remains stored in its original format and structures in a database server, typically a commercial DBMS, while access methods operate as an *add-on* facility. This requires no modification to the host database server.

Documents, manuals, and forms are part of an EIS. For keyword-based retrieval of text data, we have experimented with *signature files*. These are "fingerprints" of stored documents at about 10% of the size of the documents. Therefore, searching for all documents containing some keywords involves searching through the signature file for the fingerprints of qualifying documents. Because of the small size of the signature file, the search is very fast.

Images and text data add an order of magnitude of storage requirements and complexity. Therefore, such databases can only be stored at the database servers and remotely accessed from the workstation. However, continuous access of remote data servers is very slow, especially when one needs to do simultaneous retrieval from several images or keep switching from one to the other. To alleviate the speed problem of remote access, we use the cache techniques developed for structure databases and incremental methods for updating outdated data. Some preliminary simulations have shown that the local area network speed must be increased in order to cope with the amounts of data transfer when this involves images.

1.9. Conclusion

Our research effort during this project has focused on several areas all based on the concepts of incremental computation models. We applied the incremental models to achieve the performance required in supporting the information needs of a complex engineering environment, such as data staging, version support, update propagation, rule management, etc. Our approach in dealing with heterogeneity is that of interoperability, an approach that is accepted as the only viable solution, and we feel that we have produced a foundation for its implementation.

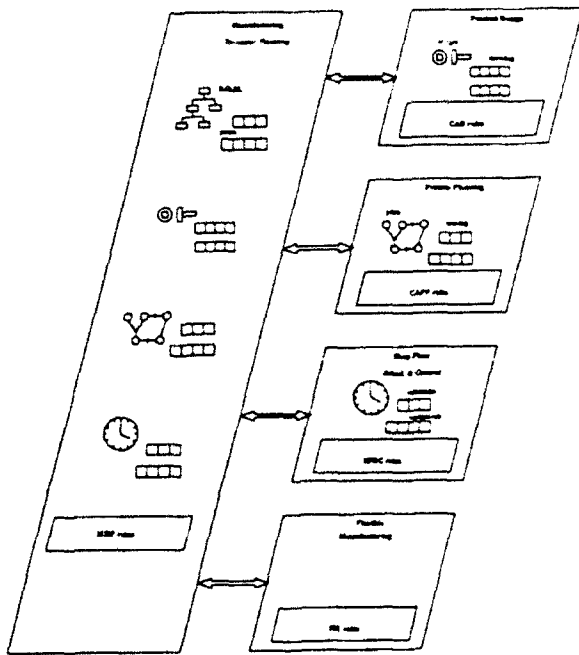


Figure 1. An Engineering Information System

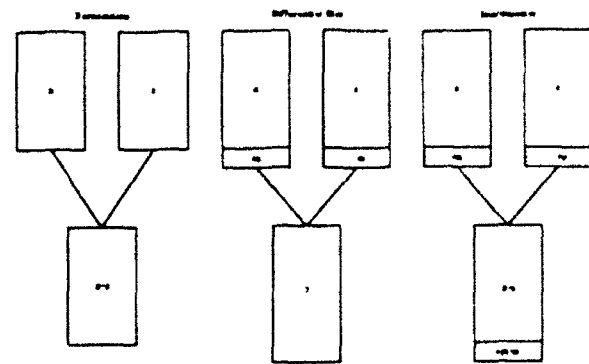


Figure 2. Incremental Computation from cached views and differential files

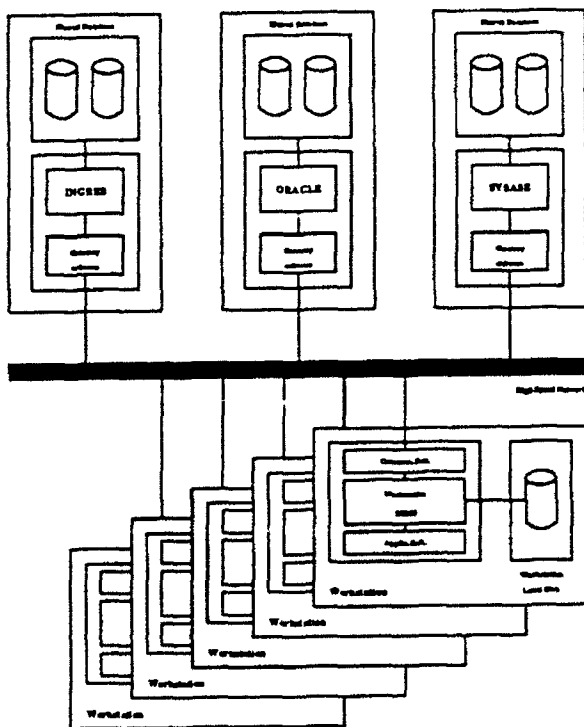


Figure 3. ADMS± Client-Server Architecture

Figure 4. ECS Architecture Throughput and Scalability.

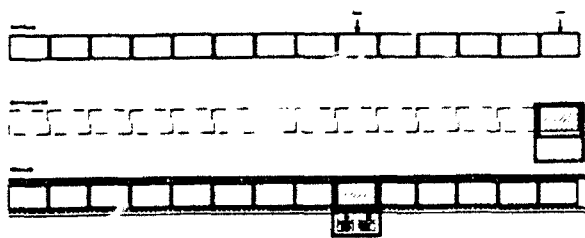


Figure 5. Conventional DBs vs. Transaction Time DBs

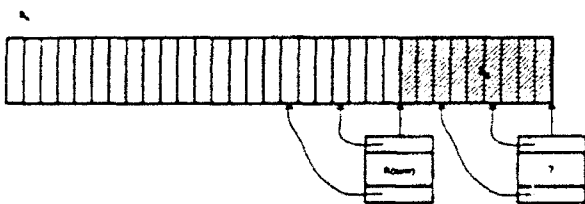


Figure 6. Incremental computation of time-slices.

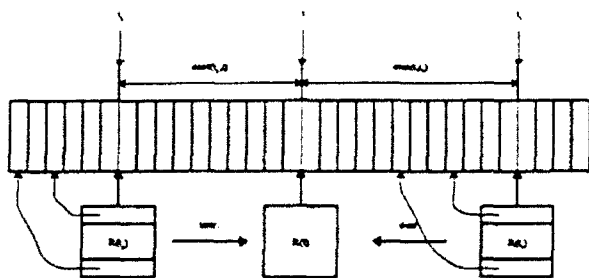


Figure 7. Differential Computation = Incremental Computation + Decremental Computation.

PART(PART#:INTEGER, NAME:STRING, UOM:UNIT)

BOM(PART#:INTEGER, SUBPART#:INTEGER)

STOCK(PART#:INTEGER, QTY:INTEGER)

CONTAINS(PART#=P, SUBPART#=SP)

→ BOM(PART#=P, SUBPART#=SP).

→ BOM(PART#=P, SUBPART#=X) and

CONTAINS(PART#=X, SUBPART#=SP)

delete PART(PART#=P)

→ PART(PART#=P) and

STOCK(PART#=P, QTY=Q) and Q>0.

write("parts in stock: no deletions made").

→ PART(PART#=P) and

STOCK(PART#=P, QTY=Q) and Q=0.

delete STOCK(PART#=P).

remove PART(PART#=P).

insert BOM(PART#=P, SUBPART#=SP)

→ CONTAINS(PART#=SP, SUBPART#=P).

write("cycle in part design").

→ ¬CONTAINS(PART#=SP, SUBPART#=P)

and ¬PART(PART#=P).

insert PART(PART#=P).

add BOM(PART#=P, SUBPART#=SP).

Figure 8. Update Dependencies controlling database updates

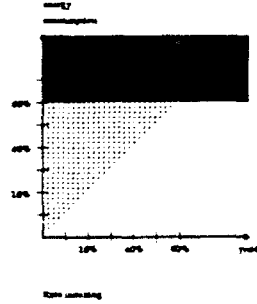
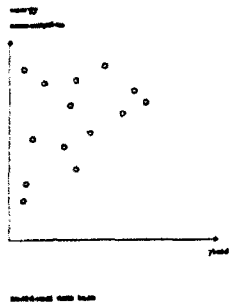
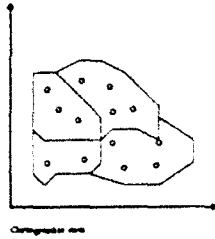
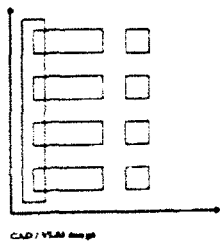


Figure 9. Other EIS Data Types

2. PART B: Detailed Research & Results

This part of the report discusses the details of the conducted research and the obtained results. The published results are cited in this section.

2.1. Summary of Conducted Research

We proposed an effective way of building High Performance Engineering Information Systems based on **Incremental Computation Models (ICMs)**. These models utilize results of previous computations stored in **persistent cache structures** by merging them with new results obtained from computations performed on the incremental changes of the database. Because computation is performed on small increments rather than the whole database, ICMs achieve performances that are far beyond the reach of ordinary systems.

During the three and a half years of the award, we have investigated a number of design issues for high performance engineering information systems. The major results of our investigation were obtained in the areas of architecture of engineering information systems, performance of incremental computation models and pipeline joins, time and version management, rule representation and performance, spatial indexing, and representation and retrieval of complex objects in engineering applications. The research has been very successful in all the proposed areas. In addition to the research, the companion development of our prototype is fairly complete- a demonstration of it was given at the SIGMOD 1990 conference. The prototype provides us with a testbed for experimenting and verifying the theoretical results.

In the following sections, we briefly describe the results published in each of these areas and lists the publications directly resulted from this grant. In total, there were 43 publications, 16 Journal articles, 7 conferences papers, 1 book chapter and 9 Technical Reports. Copies of these publications are available upon request.

2.2. Architectural Issues

We have investigated two different architectures, a tightly coupled mainframe-workstation architecture [RMSF91] and a loosely coupled multidatabase architecture [LMR91].

The mainframe-workstation architecture presented in [RMSF91] is based on **incremental computation models**. The results of queries originating from a workstation are materialized and incrementally maintained on the workstation. The architecture provides distributed functionality while ensuring high availability and low communication overhead. Explicit control of meta-knowledge is provided to support extendibility and evolution of engineering data. High levels of knowledge and activity are supported by a large rule base. Incremental computation models combined with advanced indexing and storage strategies are used to achieve high performance. Gateways to multiple relational databases and to a network database are currently under investigation.

The multidatabase system architecture presented in [LMR91] integrates multiple independently created and managed heterogeneous databases. Access is achieved without a global schema integration and each database preserves its autonomy. A formalism for the specification and enforcement of deductive and operative rules gives the system a high level of knowledge and activity and allows explicit control of interoperability in the system.

2.3. Performance Issues

We have been studying Incremental Computation Models to achieve high performance in distributed and multi-processor architectures. The results of our research on Multi-Site Database Access and Management, and Distributed and Multiprocessor Query Processing are presented in this subsection.

Incremental Access Methods in Relational Databases

We defined *ViewCache* which is a stored collection of pointers pointing to records of underlying relations needed to materialize a view. Then on top of it we developed an *Incremental Access Method (IAM)* [R91] that amortizes the maintenance cost of ViewCaches over a long time period or indefinitely. Amortization is based on *deferred* and other update propagation strategies. A deferred update strategy allows a ViewCache to remain outdated until a query needs to selectively or exhaustively materialize the view. At that point, an incremental update of the ViewCache is performed. The [R91] paper defines a set of conditions under which incremental access to the ViewCache is cost-effective. The decision criteria are based on some dynamically maintained cost parameters which provide accurate information but require inexpensive bookkeeping.

The IAM capitalizes on the ViewCache storage organization for performing the update and the materialization of the ViewCaches in an interleaved mode using one-pass algorithms. Compared to the standard technique for supporting views that requires reexecution of the definition of the view, the IAM offers significant performance advantages. We will show that under favorable conditions, most of which depend on the size of the incremental update logs between consecutive accesses of the views, the incremental access method outperforms query modification. Performance gains are higher for multilevel ViewCaches because all the I/O and CPU for handling intermediate results is avoided.

Incremental Multi-Site Database Access and Management

In [E88] and [RES92] we considered the problem of efficient multi-site database interoperation. We presented a new Client-Server architecture for databases. Incremental computation and maintenance of downloaded views were introduced and implemented.

During 1989-1990 we finished a prototype system, ADMS $\pm$ that provides *Incremental Gateway Access to Multiple Heterogeneous Database Servers*. The system's architecture allows inter-database and/or private database queries. The most attractive feature of ADMS $\pm$ is its efficiency with respect to the multidatabase management and the ability to construct *mixed-breed* relational views. The user can combine private and subsets of shared data in relational views derived from shared and private relations. This gives him the autonomy of running local SQL queries and the speed of accessing local small subsets tailored to his applications. Access to the external servers is only to obtain new data and/or for receiving incrementally the changes done shared data. The key concept in the system is the incremental maintenance algorithms offer are the foundations of the enhanced performance in such heterogeneous systems.

ADMS $\pm$'s enhanced architecture offers several advantages. First, the gateway software uses *incremental access methods* [RES92] for the interactions between the cooperating clients and servers. This reduces the number of data transfers between the workstations and the servers caused by continuous and repetitive interactions with different parts of the database or databases. Second, the local workstation environment, acting as a cache manager, becomes less dependent on the load of the servers because cached data is accessed from the local disks. Finally, most of the query processing and I/O is offloaded to powerful workstations which run in parallel. Only updates are done on the servers for maintaining consistency. However, since the ratio of updates over reads is relatively small, most of the I/O is done in parallel thus significantly increasing the I/O throughput. This was shown by extensive and thorough simulations comparing several Client-Server Database Architectures [DR92] (see also below).

Distributed and Pipelined Multiple Joins

In distributed query processing, the semijoin has been used as an effective operator in reducing data transmission and processing over a network. The semijoin is a mechanism that allows *forward* size reduction of relations and/or intermediate results used and/or generated during the processing of a distributed query. In [RK91], we propose a new relational operator, *2-way semijoin*, which enhances the semijoin with *backward* size reduction capability for more cost-effective query processing. We then introduce a Pipeline N-way Join algorithm for caching the reduced relations residing on N sites. The main advantage of this algorithm is that it eliminates the need of transferring and storing intermediate results among the sites. A worst test case

experiment is included and shows that at least 50% of the I/O is saved from the final phase of the join by eliminating intermediate results.

Multiprocessor Parallel Joins

We studied parallel execution of relational joins. Using the notion of the *Parallel Processing Tree (PPT)* that models both the parallel and the sequential aspects of the computation, we discuss in [TR89a] the optimization of parallel N-join queries on general purpose multiprocessors. The optimization is based on two basic steps: First, a cost model is derived for the PPT, and then a set of PPT operations are defined and their properties are investigated. We adopted an operational approach by iteratively applying PPT operations to transform the initial PPT into a *canonical* one which is suitable for parallel execution. The efficiency and effectiveness of this approach were justified by experiments. The application value of this optimization strategy lies in its general performance, efficiency, simplicity, generality, and flexibility.

To improve the access of data, a new index method, PB-tree, for multiprocessor environments is presented in [TR89b]. A PB-tree has a B-tree like structure but with a less rigid global structure that makes it easier to maintain in a multiprocessor environment. Each PB-tree node is formed by N subnodes, where N is the number of secondary storage devices across which the indexed relation is stored. The PB-tree has a novel node splitting method based on *k pivot keys* which are used by each processor to communicate with others. The smaller the k value, the lesser the maintenance cost is. The lowered maintenance cost is at the expense of the average index access path length. We show that moderately small pivot numbers can achieve low communication and maintenance cost with a rather insignificant loss in the average access path length. We compare the performance of the PB-tree with that using N conventional B-tree indexes each of which corresponds to a fragment of a relation partitioned on N disks. The overall performance benefits on a combined access and maintenance cost for the PB-tree over the conventional B-trees range between 22% and 34%.

Finally, the problem of *dynamic processor allocation (DPA)* at the query level in accordance with a general execution model and a general cost model is analyzed in [TR89c]. Two characterization parameters, *accumulated processor number* and *execution cost*, are used to measure the performance of a dynamic processor allocation strategy. The goal is to maximize the average parallelism by minimizing the execution cost with a given accumulated processor number. A problem variation called complete processor allocation (CPA) problem is solved in linear sequential time. The CPA has an enlarged solution domain by assuming portions of processors allocated to operations are real numbers. By discretizing the number of processors assigned to each operation, we obtain a DPA strategy called *triangle processor allocation (TPA)*. The performance of the TPA is justified and compared with other DPA strategies by experiments on sequences of random SPI_queries. The theoretical as well as experimental results show that both service demands of the operations and the execution plan structure play the key role in allocating processors to achieve high average parallelism.

Enhanced Client-Server Architecture of Databases

In [DR92] and [DR93], we examine several client-server database architectures, namely: Client-Server (CS), RAD-UNIFY type of DBMS (RU) and Enhanced Client-Server (ECS). Their specific functional components and design rationales are discussed. We use three simulation models to compare their performance under different job workloads. Our simulation results show that the RU almost always performs slightly better than the CS especially under light workloads and that ECS offers significant performance improvement over both CS and RU. Under reasonable update rates, the ECS over CS (or RU) performance ratio is almost proportional to the number of participating clients (for less than 32 clients). We also examine the impact of certain key parameters on the performance of the three architectures and finally show that ECS is scalable while the other two are not.

2.4. Time and Version Management

The database literature contains numerous papers on modeling time in relational database systems. In the past, the focus has been on data model issues and only recently has efficient implementation been addressed. In [JMR91], we present an

implementation model for the standard relational data model extended with transaction time. The implementation model exploits techniques of view materialization, incremental computation, and deferred update. It is more flexible than previously presented models based on partitioned storage. The model provides the foundation for ongoing research on version management.

Change is an often critical aspect of interest in database systems applications. A new and interesting class of queries on change behavior in the relational model extended with transaction time is presented in [JM92]. Standard relational algebra operators used on backlogs of change requests, allow for simple retrieval of change history. An extended relational algebra brings together five non-standard operators that in conjunction with the backlogs provide a powerful tool for the retrieval of patterns and exceptions in change history. Furthermore, we introduce an operator Sigma based on the notion of compact active domain. It groups data, not in predefined groups, but in groups that "fit" the data. This novel operator further expands the retrieval capabilities of the query language. The expressive power of the query language is demonstrated by examples some of which show how patterns and exceptions in change history can be detected. Sample applications of this work are statistic and scientific databases, monitoring (of production systems, databases, power plants, etc.), CAD and CASE.

A framework for query processing in the relational model extended with transaction time is presented in [JMRS93]. The framework integrates standard query optimization and computation techniques with new differential computation techniques. Differential computation incrementally or decrementally computes a query from the cached and indexed results of previous computations. The use of differential computation techniques is essential in order to provide efficient processing of queries that access very large temporal relations. Alternative query plans are integrated into a state transition network, where the state space includes backlogs of base relations, cached results from previous computations, a cache index, and intermediate results; the transitions include standard relational algebra operators, operators for constructing differential files, operators for differential computation, and combined operators. A rule set is presented to prune away parts of state transition networks that are not promising, and dynamic programming techniques are used to identify the optimal plans from remaining state transition networks. An extended logical access path serves as a "structuring" index on the cached results and contains in addition vital statistics for the query optimization process, including statistics about base relations, backlogs, about previously computed and cached, previously computed, or just previously estimated queries.

2.5. Representation and Retrieval of Complex Objects

The definition, storage, and retrieval of complex objects from relational databases is the subject of a considerable number of ongoing research projects.

In [MC92], we present a framework which provides a common foundation for the integration of databases with other areas of computer science and engineering. This framework is based on the fundamental concepts: context-free grammars and database relations.

Our goal is to provide automatic database support for complex objects which can be described by context-free grammars. Such support should include Data Definition, Data Update, Grammar Catalog Generation, Data Retrieval, and Database Restructuring. The first three of those areas are addressed in [CM89]:

- *Data Definition:* GeneRel automatically generates a set of normalized relational schemes under which objects derived from a given grammar can be stored.
- *Data Update:* GeneParse automatically generates parser specifications with insertion statements for storing sentences acceptable by a given grammar.
- *Grammar Catalog Generation:* GeneRel, when applied to a meta-grammar, generates relations in which grammars derived from the meta-grammar can be stored.

In [CM89] we present an *extended relational algebra* which can be used to retrieve information about stored grammars and stored objects derived from the grammars. We outline an algorithm, GeneView, which given a set of non-terminals from a

grammar generates a set of view definitions on the created relations.

We believe that GeneRel and GeneParse, together with our related efforts towards providing support for data retrieval and database restructuring in this environment, provide a tool which eliminates the need for manual relational database design, enhances data storage and querying, aids in the process of database restructuring, and provides a common foundation for the integration of databases with other areas of computer science and engineering.

2.6. Rule Representation and Performance

Our investigation of representation and performance has focussed on the implementation of frame-based systems, the compilation and query optimization in large rule bases, and the implementation of production systems in relational DBMS. Our accomplishments are described in the following.

A database implementation for large frame-based systems is presented in [MS89]. Frame-based systems are capable of modeling complex objects. However, many of these systems are implemented in main memory. As the number of objects to be stored far exceeds the capacity of the main memory of a computer, such an implementation is often unusable. We designed an implementation of a frame-based system on top of the POSTGRES extensible database system. Such an implementation combines the advantages of database management and frame-based systems and allows for the development of large engineering applications with minimal effort.

In [SRN90] we present our research on efficient ways to compile and store large deductive databases. The first step in such a process is to represent deductive rules using *Logical Access Paths* (LAPs). LAPs are directed graphs that capture the processing involved in the evaluation of a rule: nodes are the predicates accessed, edges indicate that a relational operation is applied on one or more nodes to derive another predicate (node). Once deductive rules are transformed to LAPs, they are integrated into a global structure, the *Rulebase Access Path Schema* (RAP Schema). The RAP Schema stores information on the inter-relationships that exist among the computations required for rule evaluations. Ground data as well as data derived through rule execution and cached for later re-use, are also modeled in the RAP Schema. Some of the advantages of using this rule base organization scheme include the automatic detection of shared computations, the inclusion of cached results of rule executions and query results in the knowledge base.

Production Systems have received a lot of attention in the context of designing and implementing rule sub-systems for relational Database Management Systems (DBMS). In particular, we have been investigating methods for storing and manipulating large rule bases using a relational DBMS. We have developed an approach to decompose and store antecedents of rules such as those used in Production Systems (PS) [SLR90]. A set-oriented approach uses a special data structure that is implemented using relations. Based on these structures, we propose a matching algorithm, similar to the Rete match algorithm of OPS5; the algorithm uses these structures to efficiently identify rule antecedents that are satisfied. In addition, we have devised a general system architecture which incorporates the proposed matching algorithm with a concurrent execution strategy for the selected set of productions [RSL90,SL90]. One advantage of the method is that it can be easily implemented on top of any relational DBMS. Actually, in order to check the feasibility of the ideas we have implemented a compiler that receives OPS5-kind of productions and generates code to implement the approach on top of the INGRES relational DBMS. Another advantage is that the method is fully parallelizable, thus making it attractive for parallel computing environments as well. In [SL90], we report preliminary results from a performance comparison of the above method versus other approaches, such as the Rete Algorithm and another proposal that suggests treating antecedents of rules as queries over the database. We define a model for rules and based on that, we present the results of a performance analysis for various database and rule base sizes and characteristics. The results, although preliminary, illustrate the effectiveness of the approach, decreasing the time required to test for applicability of productions by 50%. Finally, in [MC92], we have studied and experimented with hierarchical data structures for indexing of rule conditions.

2.7. Adaptive DBMS

In the context of Adaptive DBMSs, we have concentrated on buffer management techniques and on the design of dynamic query optimization techniques.

Previous works on buffer allocation are based either exclusively on the availability of buffers at runtime or on the access patterns of queries. In [NFS91] we propose a unified approach for buffer allocation in which both of these considerations are taken into account. Our approach is based on the notion of *marginal gains* which specify the expected reduction on page faults in allocating extra buffers to a query. In the first half of this paper, we present mathematical models and formulas computing marginal gains for various access patterns exhibited by relational database queries. Then we propose a class of buffer allocation algorithms based on marginal gains. Finally, we present simulation results that show that our approach is promising and our algorithms achieve significant improvements on throughput, as compared with previous methods.

Although in that paper we proposed the marginal gains-based policy that takes both run-time and access patterns of queries into account, and allows flexible (sub-optimal) buffer allocations to queries, this method, as well as all previously proposed buffer allocation methods, is static, that is it uses pre-calculated parameters to make the buffer allocation decisions. In [FNS91], we extend that idea in order to design an *adaptable* buffer allocation algorithm, that automatically optimizes itself for changing query workloads. The basic concept used is that of *predicting* the effect an allocation may have to the throughput of the system, and making the allocation that maximizes the expected throughput. Our preliminary results indicate that adaptable techniques outperform static ones.

In a different direction, we have also addressed issues in *dynamic* or *parametric* query optimization techniques. In most database systems, the values of many important run-time parameters of the system, the data, or the query are unknown at query optimization time. Dynamic, or parametric, query optimization attempts to identify several execution plans, each one of which is optimal for a subset of all possible values of the run-time parameters. In [INSS92] we present a general formulation of this problem and study it primarily for the buffer size parameter. We have adopted randomized algorithms as the main approach to this style of optimization and enhance them with a *sideways information passing* feature that increases their effectiveness in the new task. Experimental results of these enhanced algorithms show that they optimize queries for large numbers of buffer sizes in the same time needed by their conventional versions for a single buffer size, without much sacrifice in the output quality.

We also investigated concepts of *adaptive query optimization*. In [CR92] we studied the concept of using query execution feedback for improving database buffer management. A query feedback model which quantifies the page fault characteristics of all query access patterns including sequential, looping and most importantly random, is defined. Based on this model, a load control and a marginal gain ratio buffer allocation scheme are developed. Simulation experiments show that the proposed method is consistently better than the previous methods and in most cases, it significantly outperforms all other methods for random access.

2.8. Specification and Monitoring of Database Transactions

We have investigated two different approaches, update dependencies and transaction dependencies, to the specification and monitoring of database transactions. We describe each approach in turn.

Update dependencies are useful for specifying operational semantics of database updates in the relational model. An update dependency is a relationship between an update u_1 and a condition-update pair (c, u_2) expressing that u_1 cannot be made on a database state that satisfies c unless u_2 is also made. The set of update dependencies of an update defines a policy for transforming the current database state to a new state that has the update. This set defines an operational semantics for the update and can be used to specify application-specific knowledge that is lost when the application's data is defined in the relational model. Such application-specific knowledge primarily contains policies for maintaining database consistency, but also includes policies for maintaining and updating views.

The update dependency language [CHLM89], supports the concept of update dependencies in the relational model. For each database update, the database administrator (DBA) defines all possible correct transitions from any valid database state to one that includes the update. Each database update is transformed into an update request which is granted only if a correct transition can be found.

We have described two different strategies, depth-first and concurrent search, for executing procedures of this language, and we have developed new modified two-phase locking strategies with early write-lock release for executing these procedures in a traditional database environment.

The update dependency language has been used to specify interoperability in engineering information systems and multi-databases. The language and a prototype interpreter for it have also been used in a large-scale joint project with the Mechanical Engineering Department at the University of Maryland to validate and test the operational specification of a computer integrated manufacturing (CIM) system [HMC88,CHLM89]. From this project we learned that the update dependency language did support the specification of all the selected operations in the CIM system.

2.9. Transaction Dependency Specification

We have describe the syntax and semantics of a new declarative language TDS (Transaction Dependency Specification) in which database evolution can be specified in terms of sequential dependencies between database transactions. As an example, in a library database, we can specify that TDS specifications are regular expressions over transaction names, augmented with preconditions on the transaction history. In the library database it is possible to specify rules like 'A book must be returned within 30 days' and 'No borrower can borrow more than 10 books at a time' in terms of the involved transactions [BM91].

Transaction dependency monitoring is performed in terms of relational queries. This introduces a number of unsolved efficiency problems and we have solved some of the most important among these.

Nested queries with set membership predicates or set inclusion predicates occur naturally in TDS specifications as a notationally convenient way to express preconditions for transaction execution. We described an algorithm for transforming such queries into equivalent flat relational algebra queries. The unnesting algorithm makes it possible to utilize existing algorithms to optimize and efficiently evaluate nested queries. We have used relational algebra as both source and target language for the transformations. as a result, our unnesting algorithm is substantially simpler than previously proposed algorithms. this simplicity made it possible to formally prove the correctness of our algorithm [BM92a].

The unnesting algorithm transforms nested queries into queries that make extensive use of the relational set difference operator making the efficiency of the set difference operator an important issue. We describe algorithms for the incremental evaluation of set differences. The efficiency of the algorithms is analyzed by means of cost models and compared to reexecution algorithms. We are not aware of other work that focuses on the efficient evaluation of set differences by means of incremental methods. In many cases our algorithms are cost efficient when compared to existing reexecution algorithms [BM92b].

Time varying queries occur naturally in TDS specifications as a convenient way to express transaction preconditions that depend on time. We describe algorithms for the incremental evaluation of queries in which selection or join predicates refer to the function \$now\$. Conventional incremental methods presume that two evaluations of a given query yields the same result if no intermediate changes have been made to the underlying database. We extend the existing work on result caching and incremental evaluation and we analyze the efficiency of our algorithms and compare to reexecution algorithms. We are not aware of any other algorithms that can efficiently evaluate time varying queries incrementally. We have introduced the novel and useful notions of superviews, super caches, and predicate caches [BMC92].

2.10. References

- [BM91] Bækgaard, L., Mark, L., "Specification and Monitoring of Transaction Dependencies," UMIACS-TR-91-74, CS-TR-2678. Dept. of CS, Univ. of Md., 1991.
- [BM92a] Bækgaard, L., Mark, L., "Unnesting of Nested Algebra Queries," UMIACS-TR-92-2, CS-TR-2821. Dept. of CS, Univ. of Md., 1992.
- [BM92b] Bækgaard, L., Mark, L., "Incremental Evaluation of Time-Varying Queries Using Predicate Caches," UMIACS-TR-92-64, CS-TR-2912. Dept. of CS, Univ. of Md., 1992.
- [BMC92] Bækgaard, L., Mark, L., "Incremental Evaluation of Set Differences," UMIACS-TR-92-27, CS-TR-2851. Dept. of CS, Univ. of Md., 1992.
- [CR92] Chen, C., Roussopoulos, N., "Adaptive Database Buffer Management Using Query Feedback," submitted to 19-th *International Conference on Very Large Data Bases*, Dublin, August 1993.
- [CM89] R. Cochrane, and Mark, L., "Automating Relational Database Support for Objects defined by Context-Free Grammars - the Intension-Extension Framework," University of Maryland, Technical Report SRC-TR-89-105, December 1989.
- [DR92] Delis, A. and Roussopoulos, N., "Performance and Scalability of Client-Server Database Architectures", Proceedings of the 18-th *International Conference on Very Large Data Bases*, Vancouver, August 23-27, 1992.
- [DR93] Delis, A. and Roussopoulos, N., "Performance Comparison of Three Modern DBMS Architectures," to appear *IEEE Transactions on Software Engineering*, 1993.
- [E88] Economou, N.: "Multi-site Incremental Database Access System", MSc Thesis, Dept. of Computer Science, Univ. of Maryland, College Park, MD, December 1988.
- [FCS91] Faloutsos, C., Ng, R., and Sellis, T. "Predictive Load Control for Flexible Buffer Allocation", Proceedings of the 1991 *International Conference on Very Large Data Bases*, Barcelona, Spain, September 1991.
- [HMC88] Harhalakis, G., Mark, L., Johri, A., and Cochrane, B., "A Working Prototype MRP II/CAD System," *Computer Integrated Manufacturing Review*, vol. 5, no. 1, 1988, pp. 14-24.
- [CHLM89] Harhalakis, G., Lin, C.P., and Mark, L., "A Knowledge-Based Prototype of a Factory-Level CIM System," *Journal of Computer Integrated Manufacturing Systems*, vol. 2, no. 1, 1989, pp. 11-20.
- [INSS92] Ioannidis, Y., Ng, R., Shim, K., and Sellis, T. "Parametric Query Optimization", submitted to the 1992 *International Conference on Very Large Data Bases*, March 1992.
- [JMR91] Jensen, C. S., Mark, L., and Roussopoulos, N., "Incremental Implementation Model for Relational Databases with Transaction Time," *IEEE Transactions on Knowledge and Data Engineering*, Volume 3, number 4, December 1991.
- [JM92] Jensen, C. S. and Mark, L., "Queries on Change in an Extended Relational Model," *IEEE Transactions on Knowledge and Data Engineering*, Volume 4, number 2, April 1992, pp. 192-200.
- [JMRS93] Jensen, C. S., Mark, L., Roussopoulos, N., and Sellis, T., "Using Caching, Cache Indexing, and Differential Techniques to Efficiently Support Transaction Time," *VLDB Journal*. To appear, January 1993.
- [JM93] Jensen, C.S., Mark, L. "Differential Query Processing in Rollback Databases," in Tansel, Snodgrass, Clifford, Gardia, Segev (Eds.), *Temporal Databases*, Benjamin Cummings in association with Addison-Wesley. To appear, 1993.
- [LMR91] Litwin, W., Mark, L., and Roussopoulos, N.; "Interoperability of Multiple Autonomous Databases," *ACM Computing Surveys*, Volume 22, Number 3, September 1990, pp. 267-293.

- [MC92] Mark, L., and R. Cochrane, "Grammars and Relations," *IEEE Transactions on Software Engineering*, Vol. 18, No. 9, September 1992.
- [MRC91] "Update Dependencies in the Relational Model," (Mark, L., Roussopoulos, N., and Cochrane, R.). Submitted for journal publication October 1991.
- [MS89] Metaxas, D., and Sellis, T., "A Database Implementation for Large Frame-Based Systems," Proceedings of the Second International Conference on Data and Knowledge Systems for Manufacturing and Engineering, Gaithersburg, MD, October 1989.
- [NFS91] Ng, R., Faloutsos, C., and Sellis, T., "Flexible Buffer Allocation based on Marginal Gains", Proceedings of the 1991 ACM-SIGMOD International Conference on the Management of Data, Denver, CO, May 1991.
- [RSL90] Raschid, L., Sellis, T., and Lin, C-C., "Exploiting Parallelism in a Database Implementation of Production Systems," submitted to *IEEE Transactions on Parallel and Distributed Systems*, March 1990.
- [R91] Roussopoulos, N., "The Incremental Access Method of ViewCache: Concept and Cost Analysis," *ACM Trans. on Database Systems*, Vol. 16, No. 3, pp. 535-563, September 1991.
- [RES93] Roussopoulos, N., Economou, N., Stamenas, A., "ADMS: A Testbed for Incremental Access Methods," to appear in *IEEE Trans. on Knowledge and Data Engineering*, 1993.
- [RMSF91] Roussopoulos, N., Mark, L., Sellis, T., Faloutsos, C., "Architecture for High Performance Engineering Information Systems," *IEEE Transactions on Software Engineering*, Volume 17, number 1, January 1991, pp. 22-33.
- [RK91] Roussopoulos, N. and Kang, H., "A Pipeline N-way Join Algorithm based on the 2-way Semijoin Program," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 3, No. 4, December 1991, pp. 486-495.
- [SRN90] Sellis, T., Roussopoulos, N., and Ng, R., "Efficient Compilation of Large Rule Bases Using Logical Access Paths," *Information Systems (15)* 1, March 1990.
- [SLR90] Sellis, T., Lin, C-C., and Raschid, L., "Coupling Production Systems and Database Systems: A Homogeneous Approach," Submitted to *IEEE Transactions on Knowledge and Data Engineering*, February 1990.
- [SL90] Sellis, T., and Lin, C-C., "Performance of DBMS Implementations of Production Systems", Submitted to the 1990 Workshop on Tools for AI, May 1990.
- [TR89a] Tong, L. and Roussopoulos, N., "Iteratively Improving Parallel N-join Sequences," Institute of Advanced Computer Studies and Dept. of Computer Science, University of Maryland, Technical Report UMIACS-TR-89-116, CS-TR-2357, November 1989.
- [TR89b] Tong, L. and Roussopoulos, N., "PB-tree: A Parallel B-tree with Low Maintenance Cost," Institute of Advanced Computer Studies and Dept. of Computer Science, University of Maryland, Technical Report UMIACS-TR-89-129, CS-TR-2376, December 1989.
- [TR89c] Tong, L. and Roussopoulos, N., "Dynamic Processor Allocation for Parallel SPJ_query Execution," Institute of Advanced Computer Studies and Dept. of Computer Science, University of Maryland, Technical Report UMIACS-TR-89-117, CS-TR-2358, November 1989.

3. Student Involvement

The following students were actively involved in this project during 1988-1992. They are listed along with their area of research. Some of them were supported by this grant.

- (1) Stamenas A. (MS-graduated): Performance of Incremental Database Systems
- (2) Economou, Nicolas (MS-graduated) Multi-site Incremental Database Access
- (3) Sefrin, Eliot, PhD: Incremental Reexecution of Aborted Transactions
- (4) Tong, L. (PhD): Parallel Joins and Parallel Indexing
- (5) Metaxas, D. (PhD): Expert Database Systems
- (6) Ng, R. (PhD-Graduated): Management of Large Rule Bases
- (7) Lin, C-C. (PhD-Graduated): Performance of Production Systems coupled with and Database Systems
- (8) Cochrane, R. (PhD-Graduated): Update Dependencies; Relational Database Support for Objects defined by Context-Free Grammars
- (9) Jensen, C.S. (PhD-Graduated): Incremental Management of Transaction Time
- (10) Bækgaard, L. (PhD-Graduated) Transaction Dependencies
- (11) Delis, A. (PhD) Evaluation of Client Server Database Architectures
- (12) Chen, C. (PhD) Adaptive Query Optimizations and Buffer Management
- (13) Li, W. (PhD) Query Optimization in Client Server Database Architectures
- (14) Papakonstantinou, Y. (PhD) Computing a Query as Union of Disjoint Horizontal Fragments of Cached Views
- (15) Shim, K. (PhD) Parametric Query Optimization